Computer Updates

The Newsletter For TS Computer Owners

Volume 3, Number 3 Summer 1986

PRO/FILE CARTRIDGE UPDATE

By the time this issue of Updates reaches you, I anticipate that I'll be up to be up to my hips in Pro/File 2068 cartridge boards, eproms, and instruction books. So if you ordered one, expect it to arrive soon. Several people who had never used Pro/File 2068 asked what sort of instructions come with the new cartridge version. Here is what comes with the cartridge: 1 new copy of the Big Pro/File book; 1 copy of BREAKTHROUGH, the Pro/File addendum which shows you how to use the improved M.C. sorting facilities; and finally, a new 20 page instruction booklet which covers the new features of the cartridge such as the OR and NOT searching, use of the IBM keyboard, etc.

Unfortunately, I could not afford to cover details on the bank switching mechanism or other points on how the program works.(Booo!) That would have delayed production substantially, and the price would have to be adjusted accordingly to account for more writing and printing costs. Pro/File is already expensive enough. Rather than include this meaty information with the cartridge, I have decided to include it here in Computer Updates. (Hooray!) Many of the new features can be applied to other programs, so even if you don't get the cartridge (Boohoo!), you can still see how it all fits together.

KEYBOARD INTERFACING TIPS INSIDE!

The first chapter of the explanation of Pro/File's inner workings is inside this issue and covers the INs and OUTs of hooking up an IBM PC keyboard. What's more, the routines given here make it possible to use this keyboard with ANY Basic program. The facility is much like using a "print driver" for a big printer. You can think of this article as giving you a "keyboard driver". You load it in, and type away. When I wrote this keyboard driver, I took special precautions to insure that it would run with the cassette version of Pro/File 2068. Let me tell you, it is NICE!

NEW SYNCWARE GROUP PUBLICATION STARTING

Attention QL owners: Now you've got a magazine of your very own. "Quantum Levels" is our newest project and the first issue is due out in August. Like SyncWare News, the new "Levels" is bi-monthly. It contains product reviews, hardware projects, program listings, tips on Super Basic, and hints on using the QL bundled software. It covers the QL exclusively. There's nothing in it about other computers. Cost: $18.95 per year. Send your subscription to: Quantum Levels, P.O. Box 64, Jefferson, NH 03583.

INTRODUCTION TO CHANNELS ON THE 2068

Did you know that your 2068 has a feature built-in to it which allows you to link new peripherals into Basic? It is possible to interface your computer to modems, keyboards, printers, other computers, or just about anything else you can think of and then use Basic commands like PRINT, INPUT, LIST, INKEY, LPRINT, or LLIST to send data out or receive data from these devices.

It is the manipulation of STREAMS and CHANNELS which makes all of this possible. You probably have seen these terms mentioned before, but what they are and how to use them are topics which, until now, have been virtually ignored.

So what are streams and channels and how do they relate to external peripheral devices? Imagine opening a valve to let water pass through a garden hose. Now imagine issuing a command to send a string of characters to a modem. In the garden hose you have the water itself which is the STREAM, and you have the hose which is the CHANNEL or pathway the stream is directed through. In the modem analogy, you have the string of characters (the STREAM) which is directed through a program (CHANNEL) which outputs the stream to the modem. A STREAM is the data which is to be sent in or out. A CHANNEL is the program which directs the flow to/from the peripheral. I should emphasize, the channel is NOT the peripheral itself, but the program which handles the peripheral.

When you print characters such as the word "hello" to the TV screen you are sending a stream of data (the letters h-e-l-l-o) to a program which puts those characters on your video display. Channels are given numbers to distinguish one from the other. The channel which prints to the TV screen is channel #2. The channel which prints to your printer is channel #3. Channel #1 is the routine which prints messages at the bottom of your TV screen. When you use the basic channel commands (PRINT, LPRINT, INPUT, INKEY$, LIST, and LLIST) the computer will automatically associate the proper channel number with the command if you do not specify some different channel. Thus the command, PRINT "hello", will put the word "hello" on your TV screen.

However, you can override the default channels by specifying a different channel number. Try entering these unusual commands:

```
LPRINT #2;"HELLO"
LIST #3        (make sure your printer is on)
INPUT #2,"NAME?";#1;A$
```

You can also create your own channels in machine code and send streams of data to them. The article that follows this one shows how to connect an IBM keyboard to your computer. By creating a new keyboard channel, the commands INPUT and INKEY$ can be made to take data from this big keyboard. How is this done?

It is a two step procedure creating the channel and linking it to Basic. In the case of interfacing the IBM keyboard to the 2068, the first step is writing the software which determines which key you're pressing. Then you must link it to Basic.

The link is made by building what is called a "channel table" which tells the computer where to find the channel program. Because a stream can travel in two directions (in or out of the computer), the channel table must provide addresses for a device INPUT routine and a device OUTPUT routine. The table should also include an optional "channel identifier" which tells the computer what kind of channel it is dealing with. This identifier is simply a letter of the alphabet which signifies a device: K for Keyboard, S for screen, P for Printer, etc. You must also tell the computer where to find the channel table. This is not difficult, but it does need some explanation.

Earlier, I mentioned that there are already several channels present in the computer which handle streams from the 2068 keyboard, to the

Computer Updates is published quarterly in the winter, spring, summer and fall. Subscription price is $12.95 per year. In Canada and Mexico please add $3 postage. All other foreign subscriptions please add $9 postage. Back issues are available as single volume, 4 issue sets. Volume 1 is $9.95, volume 2 is $9.95

Edited and published by:
Thomas B. Woods
P.O. Box 64, Jefferson, NH 03583
(603) 586-7734

TV and to the 2040 printer. These channels also have a table which show where they are located. The 2068 system variable called CHANS (address 23631/2) points to the start of this table. You can look at this table by entering and running the short program below:

```
10 LET START=PEEK 23631+256*PEEK 23632
20 FOR X=START TO START+19
30 PRINT X;"=";PEEK X;TAB 12;
40 IF PEEK X<100 AND PEEK X>32 THEN
   PRINT CHR$ PEEK X:NEXT X
50 PRINT:NEXT X
```

Each channel requires 5 bytes for its table. The first 5 addresses printed on the screen represent the 2068 keyboard channel. The first 2 addresses of every channel table contain the address of the device OUTPUT routine. The second two addresses contain the address of the device INPUT routine. The fifth and last byte is used for the channel identifier. The built-in channels all have their tables together in an area of memory which begins at the address pointed to by the CHANS system variable.

When the computer needs to find an input or output routine address given in the channel tables, it does so by checking another group of 2068 system variables named "STRMS". These are 38 bytes which are described on page 262 of the TS2068 operators manual. Unfortunately, the description given in the manual is horrendously incorrect. First of all, the name, STRMS, is misleading because the 38 bytes are not "streams"; they are offsets to channels. Secondly, the book states that these bytes are "addresses of channels attached to streams." Again, they are not addresses, they are offsets.

Specifically, each two byte "STRM" tells the computer how far the channel information (the table for the specific channel) is located away from the address specified in the system variable CHANS. The value stored in a STRM will always be 1 greater than the actual distance.

Internally, the computer recognizes 7 built in channels. Their numbers are -3, -2, -1, 0, 1, 2, and 3. For all practical purposes, you can ignore these channels. Their workings are invisible to you as you run your computer. You can, however peek the first few bytes starting at address 23568 to see the offsets. Use the listing below to print them on the TV screen:

```
10 FOR X=23568 TO 23588
20 PRINT X;"=";PEEK X
30 NEXT X
```

Here is a chart showing which STRM addresses represent which channels.

STRM Adresses	Offset Value	Distance from CHANS	Channel Number
23568/69	1	0	-3
23570/71	6	5	-2
23572/73	11	10	-1
23574/75	1	0	0
23576/77	1	0	1
23578/79	6	5	2
23580/81	16	15	3
23582/83	not yet defined		4
23584/85	not yet defined		5
23586/87	not yet defined		6
23588/89	not yet defined		7
23590/91	not yet defined		8
23592/93	not yet defined		9
23594/95	not yet defined		10
23596/97	not yet defined		11
23598/99	not yet defined		12
23600/01	not yet defined		13
23602/03	not yet defined		14
23604/05	not yet defined		15

As you can see, channels 4-15 are not used internally by the computer. They are left for you to use with your add-ons. What must be done to create a new channel is to place a new channel table somewhere in memory and poke a new STRM with an offset which shows where this table is located.

You can place your 5 byte channel table just about anywhere you want as long as it is at an address higher than that specified in CHANS. I have grown fond of the practice of placing channel tables in REM lines of basic programs. This fulfills the requirement that the table be higher than the existing channel information, and if the REM holding your new channel data is the first line of a program, its location is easy to determine. The first byte of such a table will always be 5 bytes more than the address stored in the system variable, PROG (23635 and 23636).

So using the Keyboard article as an example, you locate your channel program in memory. The entry point of the keyboard software is

address FD02 hex. Then you build a channel table in a REM line. This is done by entering a program line like:

```
1 REM xxxxK
```

The address of the keyboard OUTPUT routine will be the same as the output routine for the 2068 keyboard (shown in the built-in channel table to be address 0500 hex). The first two x's must be poked with the output address (0500) and the second 2 x's must be poked with the input address (FD02).

To find the address of the first "x" in the REM line, enter the command:

```
PRINT 5+PEEK 23635+256*PEEK 23636
```

Normally, this will give 26715 as a result. So the following pokes are then entered:

```
POKE 26715,0    OUTPUT ROUTINE Equals:
POKE 26716,5          0500 HEX
POKE 26717,2    INPUT ROUTINE Equals:
POKE 26718,253        FD02 HEX
```

That takes care of the channel information. All that's left to do now is poke the first unused STRM with the proper offset. This address will depend on how many channels have already been set up. The STRM for channel 4 is at address 23582 and 23583. To determine the value to poke these addresses, enter these commands:

```
LET CHAN=PEEK 23631+256*PEEK 23632
LET TABLE=5+PEEK 23635+256*PEEK 23636
LET OFFSET=TABLE-CHAN+1
RANDOMIZE OFFSET
POKE 23582,PEEK 23670
POKE 23583,PEEK 23671
```

Now a new channel 4 is fully linked to your Basic. Using the Basic commands PRINT #4, INPUT #4, etc. will direct your stream of data to your channel software.

Describing how to make use of channels is much more difficult than actually doing it. What follows is an actual example which hopefully, you'll find far less abstract.

HOW TO CONNECT AN IBM KEYBOARD TO YOUR TIMEX 2068

This article shows you how you can use a full size IBM compatible keyboard with your computer and create a new channel to link it to your own Basic programs. My new cartridge version of Pro/File 2068 has the software part of this project built into the cartridge, and it can be used with any Basic program--not just with Pro/File. However, if you do not have the cartridge, or if you'd like to experiment with this on your TS1000, you can use this article to upgrade your cassette based programs to take advantage of a keyboard which is far better than the Timex keyboard.

The TS2068's marvelous channeling feature lets you hook the IBM keyboard right into your own Basic. This means that the INPUT and INKEY$ commands can be taught to respond to the IBM keyboard instead of the 2068 keys. Unfortunately, the TS1000 is not this flexible, however the principals for reading the keyboard are the same for both computers. I'll leave further experimenting with the TS1000 to you.

This project is fairly simple to build. All that's required besides a standard replacement keyboard for the IBM PC (such as the Keytronics Model 5151) is one of my Experimenter's I/O ports and a few miscellaneous electronic components.

If you use a program which requires a lot of input from the keyboard, you'll be amazed by the improvement this project offers. The hardware shown here can be used with the TS1000/1500 too.

General Principals:
What IS the IBM Keyboard?

Besides being a sturdy keyboard with a good feel to it, the IBM type keyboard is termed an "intelligent" keyboard because it has built-in circuitry which automatically scans the keys to see which one you press. When it finds you pressing a key, a serial steam of pulses is sent out through its data line (pin 2 of its 5 pin DIN connector) which represents a "scan code" for the key you press. This stream of electrical pulses is very similar to the stream of dits and dahs which represent a

letter in Morse Code. Each key has its own unique scan code. It is the job of the interface hardware and software to intercept the pulse stream and translate it into a character which can be understood by the computer.

A clock line comes out through pin 1 of the keyboard connector. This line also sends a serial stream of pulses at a fixed rate much like the ticks of a clock... only faster. The clock line makes it possible to "synchronize" the reading of the data line. It only ticks when data is being sent through pin 2. Each scan code consists of 8 serial pulses (or bits). Therefore, the clock ticks 8 times for each keypress.

Another characteristic of the IBM type keyboard is that each key actually produces two different scan codes depending on whether you press down on a key (commonly called the DOWN CODE) or release a key (called the UP CODE). The only difference between any given DOWN or UP code is that the 8th bit of the stream of pulses will always be low for a DOWN CODE and will always be high for an UP CODE. This means that when the serial stream is converted into an 8 bit number, an UP CODE for a specific key will always be 128 (that's 80 hex) more than the corresponding DOWN CODE. Because of this, it is very easy for the software to tell if you are pressing down on a key or if you have released it (pressing up?). If the scan code is less than 128, your finger is on a key. If it is greater than 128 you have released it.

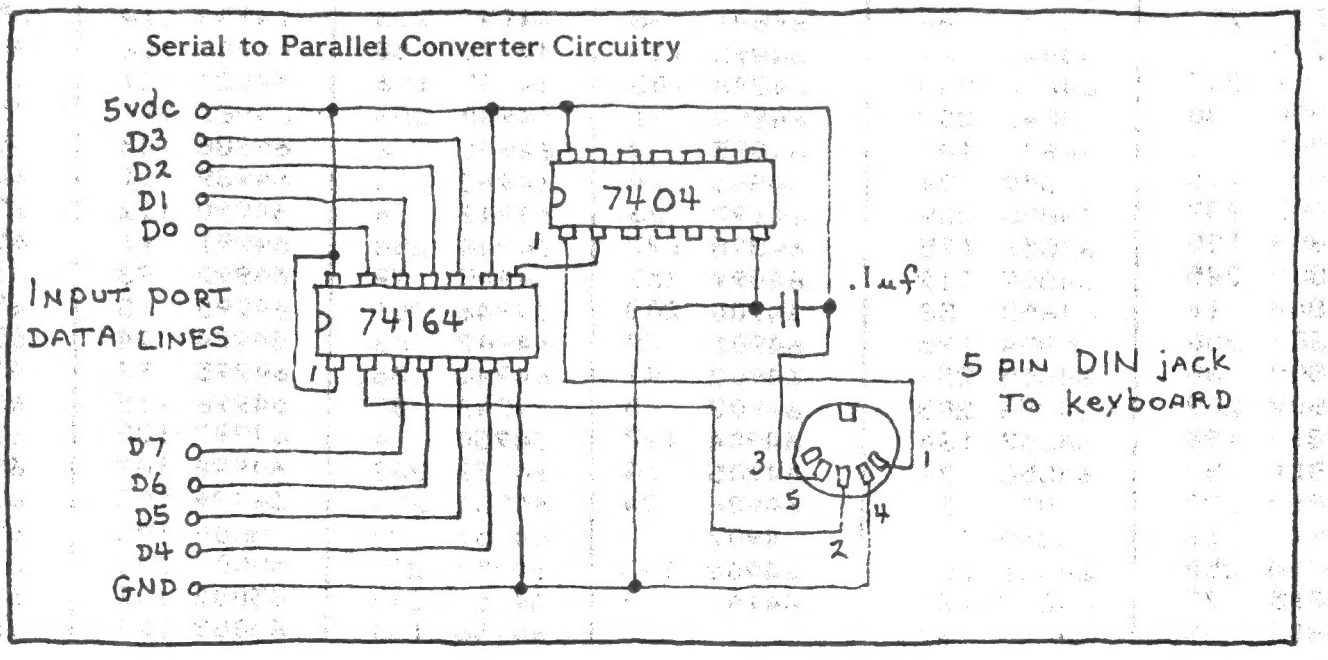
The last important feature of the keyboard is that it will automatically send repetitive scan codes if you press a key and hold it down. While this bit of "intelligence" may work just fine on an IBM computer, it turns out to be a major headache for the Timex, but the keyboard software gets around this problem and still manages to provide for auto repeating keys.

Interface Hardware Requirements

In order for the computer to make sense out of the serial stream of keyboard data, some means of converting it into a parallel byte is necessary. The electronic circuitry shown here does this. The schematic shows the keyboard data and clock lines on pins 2 and 1 of the DIN connector respectively. The serial data stream is fed to pin 2 of the 74164 serial to parallel converter chip. The clock is inverted (highs become lows and lows become highs) by the 7404 and then is sent to the clock input pin (pin 8) of the 74164. These two lines are all that is necessary for the 74164 to convert the stream into 8 parallel bits. These outputs (D7 through D0) must then be connected to their respective places on the IN/OUT board's input lines.

The power supply lines (+5v and Gnd) necessary to run both the converter and the keyboard can be taken directly from the I/O Port.

Wiring this circuit is easy. There's nothing critical about the way lines are layed



out. Just make sure all the connections go to the right places.

Whenever I build circuits like this, I like to do them on solderless bread boards which are available at Radio Shack. Once I have everything working properly, I take it apart and rebuild it on perfboard with soldered connections.

Once you get your circuit built, connect it to your computer and test it out. Enter the test listing (the Basic line 10 which immediately follows the Keyboard table) on the next page.

Table 1. IBM Keyboard Software Pokes

Address	Value	Address	Value	Address	Value	Address	Value	Address	Value	Address	Value
64770	205	64817	194	64864	225	64910	126	64957	24	65004	59
64771	30	64818	253	64865	33	64911	50	64958	144	65005	39
64772	253	64819	254	64866	195	64912	196	64959	14	65006	39
64773	121	64820	170	64867	253	64913	253	64960	255	65007	0
64774	12	64821	40	64868	254	64914	78	64961	201	65008	92
64775	32	64822	15	64869	70	64915	6	64962	0	65009	122
64776	2	64823	254	64870	40	64916	0	64963	0	65010	120
64777	207	64824	182	64871	87	64917	58	64964	0	65011	99
64778	12	64825	40	64872	254	64918	194	64965	0	65012	118
64779	167	64826	11	64873	42	64919	253	64966	199	65013	98
64780	202	64827	254	64874	40	64920	201	64967	49	65014	110
64781	14	64828	184	64875	50	64921	33	64968	50	65015	109
64782	12	64829	40	64876	254	64922	120	64969	51	65016	44
64783	33	64830	20	64877	54	64923	254	64970	52	65017	46
64784	59	64831	190	64878	40	64924	24	64971	53	65018	47
64785	92	64832	32	64879	46	64925	230	64972	54	65019	0
64786	203	64833	26	64880	254	64926	203	64973	55	65020	203
64787	238	64834	1	64881	58	64927	206	64974	56	65021	0
64788	50	64835	0	64882	40	64928	24	64975	57	65022	32
64789	8	64836	0	64883	63	64929	160	64976	48	65023	0
64790	92	64837	201	64884	254	64930	203	64977	45	65024	0
64791	55	64838	175	64885	59	64931	198	64978	61	65025	205
64792	195	64839	119	64886	40	64932	24	64979	12	65026	226
64793	14	64840	33	64887	46	64933	156	64980	9	65027	7
64794	12	64841	195	64888	254	64934	175	64981	113	65028	4
64795	209	64842	253	64889	56	64935	50	64982	119	65029	8
64796	16	64843	203	64890	40	64936	194	64983	101	65030	10
64797	253	64844	86	64891	38	64937	253	64984	114	65031	11
64798	1	64845	32	64892	126	64938	58	64985	116	65032	9
64799	207	64846	243	64893	203	64939	196	64986	121	65033	15
64800	80	64847	203	64894	71	64940	253	64987	117	65034	0
64801	96	64848	142	64895	32	64941	6	64988	105	65035	0
64802	175	64849	24	64896	24	64942	0	64989	111	65036	55
64803	237	64850	239	64897	33	64943	16	64990	112	65037	56
64804	120	64851	175	64898	197	64944	254	64991	91	65038	57
64805	245	64852	119	64899	253	64945	79	64992	93	65039	45
64806	16	64853	33	64900	203	64946	201	64993	13	65040	52
64807	250	64854	195	64901	79	64947	203	64994	94	65041	53
64808	68	64855	253	64902	40	64948	86	64995	97	65042	54
64809	209	64856	203	64903	5	64949	32	64996	115	65043	43
64810	186	64857	134	64904	197	64950	4	64997	100	65044	49
64811	32	64858	24	64905	14	64951	203	64998	102	65045	50
64812	239	64859	230	64906	84	64952	214	64999	103	65046	51
64813	16	64860	119	64907	9	64953	24	65000	104	65047	48
64814	250	64861	203	64908	193	64954	227	65001	106	65048	46
64815	79	64862	127	64909	9	64955	203	65002	107	65049	0
64816	33	64863	32			64956	150	65003	108	65050	199

Address	Value	Address	Value	Address	Value
65051	33	65082	70	65113	8
65052	64	65083	71	65114	10
65053	35	65084	72	65115	11
65054	36	65085	74	65116	9
65055	37	65086	75	65117	15
65056	94	65087	76	65118	0
65057	38	65088	58	65119	0
65058	42	65089	34	65120	55
65059	40	65090	39	65121	56
65060	41	65091	0	65122	57
65061	95	65092	92	65123	45
65062	43	65093	90	65124	52
65063	12	65094	88	65125	53
65064	9	65095	67	65126	54
65065	81	65096	86	65127	43
65066	87	65097	66	65128	49
65067	69	65098	78	65129	50
65068	82	65099	77	65130	51
65069	84	65100	60	65131	48
65070	89	65101	62	65132	46
65071	85	65102	63	65133	198
65072	73	65103	0	65134	11
65073	79	65104	203	65135	197
65074	80	65105	0	65136	45
65075	123	65106	32	65137	8
65076	125	65107	0	65138	0
65077	13	65108	0	65139	9
65078	94	65109	205	65140	43
65079	65	65110	226	65141	195
65080	83	65111	7	65142	10
65081	68	65112	4	65143	195

```
10 PRINT AT 1,0;IN 223;"   ": GO TO 10
```
note: 3 spaces between the quotes

Press different keys on the IBM keyboard. Line 10 will continuously read the keyboard port and print the keyboard scan code. Notice that each key produces its own unique pair of UP codes and DOWN codes, and that the UP code will equal the DOWN code plus 128.

Keyboard Software

To use the IBM keyboard with your 2068, poke the values shown in table 1 into addresses 64770 through 65143. Use your favorite loader program to do this or enter and run the simple loader shown below:

```
10 CLEAR 64769
20 FOR X=64770 TO 65143
30 INPUT "VALUE FOR ";(X);"?";Y
40 POKE X,Y: PRINT X;"=";Y
50 NEXT X
```

After all the bytes have been poked, save it off to tape with the command,

```
SAVE "IBM"CODE 64770,375
```

Now, any time you want to use the software, load it into your computer by using the command,

```
LOAD "IBM"CODE
```

Then add these lines to the basic program you wish to use the IBM keyboard with:

```
   1 REM xxxxK
   2 POKE 26715,0: POKE 26716,5:
POKE 26717,2: POKE 26718,253: P
OKE 23582,28
```

Line 1 is the "channel table". It is vitally important that it be the first line of your basic program. This software also assumes that your program starts at the normal starting address of 26710 decimal. Line 2, pokes the "x's" in line 1 with the values necessary so the computer will know how to find the keyboard software. The first two pokes are the address of the channel OUTPUT routine. The second two pokes are the address of the channel INPUT routine--the address of our keyboard software. The last poke which goes to the streams table gives the distance from the new keyboard channel table to the start of normal channel table. You can change line 2's line number if necessary, but the line MUST be executed BEFORE you try to read the IBM keyboard.

And how do you read the keyboard? Well, the keyboard can be read by any Basic Input or Inkey$ command which specifies channel #4. For example:

```
INPUT #4;"MAKE YOUR SELECTION";A$
LET Y$=INKEY$#4
INPUT "HOW MUCH";#4;X
```

Therefore, what you must do is go through your basic listing and change every occurence of INPUT and INKEY$ to INPUT #4 and INKEY$#4.

The IBM keyboard will function only when the program is being run. When you stop to edit program lines you must use the 2068 keyboard.

The INPUT #4 command accepts keypresses from both the IBM and the 2068 keyboards. The

INKEY$#4 command is different in that it reads only the IBM keyboard. If you want to make it possible to read BOTH keyboards using INKEY$, enter a program line something like this:

```
LET K$=INKEY$+INKEY$#4:IF LEN K$>1
THEN LET K$=K$(1)
```

This line will read both the 2068 keyboard and the IBM keyboard into k$. If no keys are pressed, or if just one keyboard or the other is being used, K$ will be empty or will equal the character being pressed. If you press keys on both keyboards simultaneously, K$ will assume the value of the 2068 keyboard only.

Features of the Keyboard Software

All standard ASCII characters (letters, numbers, and punctuation marks) work according to the symbol printed on the key. There are 10 special function keys which I have pre-defined. Because this software was written primarily for use with Pro/File 2068, these function keys are set to perform various functions within that program.

F1-Auto Repeat. Hold this key down to repeat the last keypress.
F2-STEP Token (Step to next file in Pro/File)
F3-STOP token (Close file in Pro/File)
F4-CHR$ 7 (INSERT/OVER switch in Pro/File)
F5-CHR$ 4 (Line Delete in Pro/File)
F6-Cursor LEFT
F7-Cursor DOWN
F8-Cursor UP
F9-Cursor RIGHT
F10-CHR$ 15 (More Commands in ADD/EDIT mode)

The shift and caps lock keys work as you would expect. Holding shift and some other key will produce upper case. The caps lock key is an upper/lower case switch. Press it once to make all keypresses upper case. Press it again to make all keypresses lower case.

Defining Alternate Keys on the IBM

The ALT key is an ALTernate shift key, however its function is not to produce upper case characters. ALT can be pressed along with some other key to produce your own custom defined character. At the start, pressing ALT and some other key will have no effect because you haven't defined any keys yet. Here's how to do it.

The table below shows memory addresses which store the key code to be used when you press ALT with some other key. If you poke a key address with the code for the character you wish the key to represent, you can obtain that character when you simultaneously press ALT and the desired key.

Address	IBM key	Address	IBM Key
65145	ESC	65187	\
65146	1	65188	z
65147	2	65189	x
65148	3	65190	c
65149	4	65191	v
65150	5	65192	b
65151	6	65193	n
65152	7	65194	m
65153	8	65195	,
65154	9	65196	.
65155	0	65197	/
65156	-	65198	not used
65157	=	65199	Pr Scr
65158	Back Sp	65200	not used
65159	TAB	65201	Space
65160	q	65202	not used
65161	w	65203	not used
65162	e	65204	F2
65163	r	65205	F3
65164	t	65206	F4
65165	y	65207	F5
65166	u	65208	F6
65167	i	65209	F7
65168	o	65210	F8
65169	p	65211	F9
65170	[	65212	F10
65171	]	65213	NUM LOCK
65172	RETURN	65214	not used
65173	CTRL	65215	HOME
65174	a	65216	Cur UP
65175	s	65217	Pg up
65176	d	65218	-
65177	f	65219	Cur LEFT
65178	g	65220	Alt 5
65179	h	65221	Cur RIGHT
65180	j	65222	+
65181	k	65223	End
65182	l	65224	Cur DOWN
65183	;	65225	Pg DOWN
65184	'	65226	Ins
65185	`	65227	Del
65186	not used		

The addresses on the next page give keys in UPPER CASE mode. Access them by pressing SHIFT and ALT simultaneously with one of the other keys.

Address	IBM Key	Address	IBM Key
65229	ESC	65271	I
65230	!	65272	Z
65231	@	63573	X
65232	#	65274	C
65233	$	65275	V
65234	%	65276	B
65235	^	65277	N
65236	&	65278	M
65237	*	65279	<
65238	(	65280	>
65239	)	65281	?
65240	_	65282	not used
65241	+	65283	PR SCR
65242	Back Sp	65284	not used
65243	TAB	65285	SPACE
65244	Q	65286	not used
65245	W	65287	not used
65246	E	65288	F2
65247	R	65289	F3
65248	T	65290	F4
65249	Y	65291	F5
65250	U	65292	F6
65251	I	65293	F7
65252	O	65294	F9
65253	P	65295	F9
65254	{	65296	F10
65255	}	65297	NUM LOK
65256	RETURN	65298	not used
65257	CTRL	65299	HOME
65258	A	65300	CUR UP
65259	S	65301	PAGE UP
65260	D	65302	-
65261	F	65303	CUR LEFT
65262	G	65304	5
65263	H	65305	CUR RIGHT
65264	J	65306	+
65265	K	65307	END
65266	L	65308	CUR DOWN
65267	:	65309	PG DOWN
65268	"	65300	INSERT
65269	~	65301	DELETE
65270	not used		

Here is an example of how you might define a key. Say you want to make it possible to produce the AND token whenever you press ALT and F10 together. What you must do is determine the code for the AND token. You'll find the code listed on page 244 of the 2068 owner's manual. The table printed there shows that the number 198 is the code for AND. Now refer to the table above and change the address representing F10 so that it stores the value 198. Enter the command:

POKE 65212,198

to change the value representing the F10 key. Now, any time you press F10 and ALT together, you will get the token AND. This works as long as you are in lower case. If you want F10 to respond the same way in upper case as well, you must also POKE 65296 with a 198.

IBM Keyboard Disassembly

```
FD02 CD1EFD    KEY? CALL IBM$
FD05 79             LD A,C
FD06 0C             INC C
FD07 2002           JR NZ,FD0B
FD09 CF             RST 08H
FD0A 0C              Error D
FD0B A7             AND A
FD0C CA0E0C         JP Z,0C0E
FD0F 21385C         LD HL,5C3B
FD12 CBEE           SET 5,(HL)
FD14 32085C         LD (5C08),A
FD17 37             SCF
FD18 C30E0C         JP 0C0E
FD1B D1        STAK POP DE
FD1C 10FD      CLER DJNZ,STAK
FD1E 01CF50    IBM$ LD BC,50CF
FD21 60             LD H,B
FD22 AF        SAMP XOR A
FD23 ED78           IN A,(C)
FD25 F5             PUSH AF
FD26 10FA           DJNZ,SAMP
FD28 44             LD B,H
FD29 D1             POP DE
FD2A BA             CP D
FD2B 20EF           JR NZ,CLER
FD2D 10FA           DJNZ,FD29
FD2F 4F             LD C,A
FD30 21C2FD         LD HL,LSCN
FD33 FEAA           CP AA
FD35 280F           JR Z,UNSH
FD37 FE86           CP 86
FD39 280B           JR Z,UNSH
FD3B FE88           CP 88
FD3D 2814           JR Z,UNAL
FD3F BE             CP (HL)
FD40 201A           JR NZ,CHAR
FD42 010000    QUIT LD BC,0000
FD45 C9             RET
FD46 AF        UNSH XOR A
FD47 77             LD (HL),A
FD48 21C3FD         LD HL,KFLG
FD4B CB56           BIT 2,(HL)
FD4D 20F3           JR NZ,QUIT
FD4F CB8E           RES 1,(HL)
FD51 18EF           JR QUIT
```

Keyboard Disassembly Cont.

```
FD53 AF         UNAL XOR A
FD54 77              LD (HL),A
FD55 21C3FD          LD HL,KFLG
FD58 CB86            RES 0,(HL)
FD5A 18E6            JR QUIT
FD5C 77         CHAR LD (HL),A
FD5D CB7F            BIT 7,A
FD5F 20E1            JR NZ,QUIT
FD61 21C3FD          LD HL,KFLG
FD64 FE46            CP 46
FD66 2857            JR Z,BRAK
FD68 FE2A            CP 2A
FD6A 2832            JR Z,SHFT
FD6C FE36            CP 36
FD6E 282E            JR Z,SHFT
FD70 FE3A            CP 3A
FD72 283F            JR Z,CAPS
FD74 FE3B            CP 3B
FD76 282E            JR Z,REPE
FD78 FE38            CP 38
FD7A 2826            JR Z,ALT_
FD7C 7E         DCOD LD A,(HL)
FD7D CB47            BIT 0,A
FD7F 2018            JR NZ,SPEC
FD81 21C5FD          LD HL,TBL0
FD84 CB4F            BIT 1,A
FD86 2805            JR Z,LOWR
FD88 C5         UPPR PUSH BC
FD89 0E54            LD C,54
FD8B 09              ADD HL,BC
FD8C C1              POP BC
FD8D 09         LOWR ADD HL,BC
FD8E 7E              LD A,(HL)
FD8F 32C4FD          LD (LASK),A
FD92 4E              LD C,(HL)
FD93 0600            LD B,00
FD95 3AC2FD          LD A,(LSCN)
FD98 C9              RET
FD99 2178FE     SPEC LD HL,TBL1
FD9C 18E6            JR FD84
FD9E CBCE       SHFT SET 1,(HL)
FDA0 18A0            JR QUIT
FDA2 CBC6       ALT_ SET 0,(HL)
FDA4 189C            JR QUIT
FDA6 AF         REPE XOR A
FDA7 32C2FD          LD (LSCN),A
FDAA 3AC4FD          LD A,(LASK)
FDAD 0600            LD B,00
FDAF 10FE            DJNZ,FDAF
FDB1 4F              LD C,A
FDB2 C9              RET
FDB3 CB56       CAPS BIT 2,(HL)
FDB5 2004            JR NZ,FDBB
FDB7 CBD6            SET 2,(HL)
FDB9 18E3            JR SHFT
FDBB CB96            RES 2,(HL)
FDBD 1890            JR FD4F
FDBF 0EFF       BRAK LD C,FF
FDC1 C9              RET
FDC2 00         LSCN NOP
FDC3 00         KFLG NOP
FDC4 00         LASK NOP
FDC5 00         TBL0 NOP
```

PRO/FILE BITS & PIECES

Charles Stelding from Tyler, TX writes... "Pro/File 2068 makes an excellent calendar and daily schedule. I use it in my work as a Campus Minister at Tyler Junior College. Here is a print out showing how I use the calendar.

```
        APRIL 1985
--------------------

 S  M  T  W  T  F  S    TO READ
                        CALENDAR
    1! 2! 3  4  5! 6    TYPE DATE
                        "APRIL 1"
 7  8! 9!10 11 12 13

14 15!16!17 18 19 20

21 22 23!24 25 26!27!

28 29 30!31
--------------------
```

I place a "!" after each day that something is scheduled which allows me to view the entire month. You can then view the specific day of things scheduled by typing in the month and day.

```
APRIL 2

 9:00 Work up calendar schedule
      for next year to hand in
      to Ms. Pratter.
10:00 Get things finalized for
      the Rel. Emphasis week
      a. Check school calendar
      b. Call McCaghren
      c. Line up singers
11:30 Student lunch

 2:00 Table talk
```

Note: Although this application is used on Pro/File 2068, you can do the same thing with ZX Pro/File for the TS1000

And from B.R. Downey, E. Lansing, MI... "How can I take the asterisk off of the first line in ZX Pro/File when I make mailing labels?"

Unfortunately, it is absolutely impossible to remove the asterisk completely without a major rewrite of the program. However, you can avoid printing the "*" when making mailing labels. The asterisk is present only on the first line of any record. It is printed, therefore, only when you try to print the first line. If you arrange your data so that a name and address starts on line 2, you can set your DEFP accordingly so that the first line with the "*" will not be printed.

From Robert Hartung, Huntertown, IN "...Pro/File 2068 owners who are using Spectrum ROMs or emulators will be interested to know that the Pro/File print driver requires just two pokes to make it work with the Spectrum ROM. The Call address for the break scan needs to be changed from 63688=9 and 63689=32 in the original (Pro/File) location to 63688=84 and 63689=31 for the Spectrum ROM. After doing this, the Pro/File adapted to the Spectrum will Lprint perfectly."

Editor's note: These addresses can be found in the POKER TABLE on page 118 of the "Big Book". While on the subject of Spectrum compatibility, Pro/File 2068 must get 2 pokes to its machine code and some of its Basic program lines changed in order for it to run in the Spectrum. Addresses 63666 and 63667 you must poke to zero. All ON ERR commands must be deleted. The easiest way to find them is to run the program in a spectrum and wait for the computer to stop with a "Nonsense in Basic" error report. Then list and edit the offending line, deleting the parts which cause the error. Removing these commands causes no ill effect on the program's operation other than the fact that the machine may stop with an error if you try to input faulty information or if you press BREAK while in the ADD/EDIT mode.

A few thoughts from Basil Wentworth, Bloomington, IN ...Applying Pro/File 2068 to my index of music manuscripts, I can never remember whether to search for Tuba, or TUBA, or tuba, so I added the line:

```
10 POKE 23658,8
```

which puts everything in capital letters. This also makes it easier to enter the key letters called for when I'm building a file.

Second, I added line:

```
60 PRINT TAB 4;""""INDEX gives Subjects""""
```

Then I make one entry into the file something like the following:

INDEX TITLE COMPOSER INSTRUMENT(S) TYPE FILE LOCATION etc etc

and one of my records would look something like:

THE AERIALIST BASIL TUBA AND PIANO/BAND BRAVURA HUMOROUS CIRCUS RED FOLDER

and maybe some information about where I have already performed the piece. Now, by searching for "INDEX" first, I can find whether there's any point in looking for BRAVURA, or whether I have to try VIRTUOSO or TOUR DE FORCE or FLASHY AND TRASHY or whatever.

Editors note: The use of an INDEX file or HELP file is a very useful aid which works in both the TS1000 and the 2068 versions of Pri/File. In my own use of the programs to manage mailing lists, I have several different programs and each one uses different lines for a person's name and address. In each program, I have added a file which tells where the name, street, etc is to be located. I also put the word "HELP" somewhere within this file. Then when I need a reminder to tell me what line to put my data on, I type "HELP" as a search command. The computer will then find and display my HELP file and I have my reminder. It saves hours of fustration.

****

32K NON-VOLATILE MEMORY UPDATES

Here is a simple modification you can make to the NVM which enables you to remove an old battery and plug in a new one WITHOUT losing the program you have stored in it! It is amazing, but true. After removing the battery, you can diddley-bop around for 5 minutes (!) and still have time to plug in a new battery.

This mod is so good, we have started including it with all new RAMs we ship. However, if you have an older model, you can add it yourself by following the instructions below:

1. Locate the small capacitor which is immediately to the right of the write/protect switch.

2. If the capacitor is an electrolytic "can", go no further. Your NVM is already fitted.

3. If the capacitor is a small tear-drop shape, remove it with a grounded soldering iron (the 3-prong type).

4. Replace it with a 220 uf, 16 volt electrolytic capacitor (Radio Shack #272-956). The negative lead of the cap should be connected to the hole which connects to the big fat ground trace. This is the hole which is located closest to the "DK" legend printed next to the Dock/Exrom Switch.

I find it simply amazing that this capacitor can keep memory alive for 5 minutes without any battery connected. It is testament to the infinitesimal current drain of the NVM, that a capacitor can hold the ram for so long.

****

HOT Z CLINIC

Last time, we looked at the details surrounding HOT-Z's READ mode. This issue, we'll use HOT-Z to write some machine code. The program we write will not be a masterpiece that sends shivers up your spine. Instead, it will be something short and sweet, but it will demonstrate the way you use HOT-Z to write code. What I have chosen is a machine code routine which will tell you how many bytes of free memory you have which can be used for additional Basic program lines and variables. The TS2068 already has this function built into its Basic language. This is the FREE function. When you execute the command PRINT FREE, it prints out the number of "free" bytes which are still unused. All you 2068 owners, please bear with me as we go through the examples. The intent here is to learn how to use HOT-Z, not to receive spoon-fed programs.

Load HOT-Z into your computer, and work your way to the disassembly display in READ mode as shown in the last issue. The first step in writing machine code with HOT-Z is deciding where in memory our program should be located. The object is to pick a convenient place which will not be get in the way of the Basic operating system. Machine code programs can be placed just about anywhere, but if you don't take special precautions, the Basic operating system will not intuitively know where it is. As a result, your machine code can be erased accidently without you even knowing about it. The Basic is like an elephant and your machine code program is an ant. You must be careful not to be trampled.

Fortunately it is very easy to set aside an area of memory and tell the Basic system to leave it alone. The time honored tradition in the TS1000 is to create a REM line in basic which contains enough blank spaces to hold your code. As you shall see, this can also be done on the 2068. When you enter a Basic REM line which contains some number of letters or spaces after the rem token, you are using the operating system itself to set aside a block of memory as a basic program line. The characters or spaces that you start with can then be changed into machine code instructions.

Another way to set aside memory for M.C. programs is to tell the computer there is less memory attached than what there really is. This technique is called "lowering ramtop". Using this technique, the highest addresses are made unavailable to the Basic. As more program lines and variables consume memory, the computer will eventually reach a point beyond which nothing more can be added. It will stop with an "Out of Memory" error. In a computer with a lowered ramtop, the computer will run out of memory before reaching the actual top of ram. The bytes you set aside by lowering ramtop are safely preserved and will not be accidently overwritten.In our program, we shall choose the first option of reserving bytes: creating a REM line.

Going from HOT-Z to Basic

To do this, use the QUIT command to turn HOT-Z off. On the TS1000, press SHIFT and the letter "Q". On the 2068, press Symbol Shift and the letter "Q". The result is the same in both computers: HOT-Z is turned off and the Basic operating system is restored.

Now enter a rem line with 50 "X"'s following the REM token. After the line is entered, the screen should look like:

```
    1 REM XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXX
```

Going From Basic to HOT-Z

Now we can go back into HOT-Z and change those "X"'s into real machine code commands. If you use the TS1000, enter the command RAND USR 22528. To turn the 2068 HOT-Z back on, enter RANDOMIZE USR 24098. This puts us right back in HOT-Z's READ mode. The next task is to use HZ to find the rem line we just entered.

Find the Basic Program With HOT-Z

This is simple on the TS1000. The owner's manual states that All basic programs start at the address 16509 in decimal. The 2068 is a bit different in that Basic programs need not always start at the same address. Different hardware or software configurations may cause a program to start in a different position. There is, however a 2068 system variable called "PROG" which will always hold the address where the Basic program begins (see page 263 of the 2068 owner's manual). PROG is a two byte variable at address 23635 and 23636.

Read Memory by Inputting a Decimal Address

In the last issue we talked about displaying a specific group of addresses by typing in a 4 digit hex address to make HOT-Z display a screenful of bytes starting at the address you type in. HOT-Z includes a very nice feature which lets you type in the starting address in decimal too. This is very useful in our case where we need to jump to decimal 16509 in the TS1000 or decimal 23635 in the 2068.

If you use the TS1000, type SHIFT and "3". 2068 owners should press Symbol Shift and "U". This tells HOT-Z to expect a 5 digit decimal number rather than a 4 digit hex. After pressing these keys, the top left of the screen will change (this is where it says "ADDR") to make room for a 5 digit input.

Now you can type in an address in decimal. For the TS1000, type 16509 to make the computer show you the start of the Basic Program. For the TS2068, type 23635. This will show you the values of some of the 2068 system variables. The top two bytes will be the address of the start of the Basic program. The 2068 will show address 5C53 holding the value of 56 and the address 5C54 holding the value 68. One might deduce, therefore that the address of the Basic is 5668 hex, but that is incorrect. All computers store numbers in the exact opposite order that humans do. To the computer, a 56 followed by a 68 translates to the hex address of 6856. Now that we have looked up the address in PROG, we can make HOT-Z display this area in memory simply by typing the address in. So type: 6856. At this point, both in the TS1000 and the 2068, HOT-Z should be READING a disassembly of the start of your Basic REM line.

It doesn't look much like a rem line does it? This is because Basic program lines are NOT machine code, they are data which is interpreted by the ROM. Since the computer is in the disassembly mode right now, it is trying to disassemble data. The result is a nonsensical display. You may recall that we covered this type of situation in the last issue Refer to it for more details. To view the program line properly, we must go to the DATA MODE. On the 2068, press Symbol Shift and "G". On the 1000 press Shift and "D". Do you see the REM token and the "X"'s which make up your program line now? The REM token should be found on the fifth line of the data display. Below it, you will find the first screenful of X's which you typed in. You may be wondering what the top four lines of the data display represent. These are the addresses which hold the decimal values 0, 1, 52, and 0 respectively. I refer you to your computer manual--the chapter entitled "Organization of Storage". On the TS1000 page 128 and page 255 in the 2068 book, you will find a nice chart which depicts the byte structure of a Basic program line. As the chart says, the first 2 bytes represent the line number (the HOT-Z display indicates that we must be looking at a line number 1), the second two bytes show the length of text plus the ENTER character which is always the last character of any program line. HOT-Z shows this line has a length of 52 bytes (that's one byte for the REM token, 50 X's, and 1 byte for the ENTER character. 1+50+1=52).

Generally, you should not try to change these first four "pointers" of the program line unless you know what you're doing. We'll cover that at a later date, however, I will mention that HOT-Z can put whatever values you choose into these pointers. Changing the first two bytes will change the line number when you go back to Basic and list the program. Changing the second two bytes will do all kinds of crazy things. Many of them will cause your computer to crash the instant you go back to basic. The TS1000 version of MTERM (modem software) had these bytes changed as a copy protection scheme. Pro/File Updates Vol.2 No.4 (pg.2-3) showed how to merge two smaller rem lines into 1 big one by changing these two length pointers.

For our present task, we want to find the address of the first "X" after the REM token. This is where we are going to begin writing our machine code. Before going any further write down both the hex and decimal addresses of this "X". On the TS1000 it will be 4082 hex and 16514 decimal (does that number ring a bell?). On the TS2068 you should write down 685B hex and 26715 decimal.

Writing Machine Code

Now switch back to the DISASSEMBLY MODE (SHIFT and "D" for TS1000, or Symbol Shift and "G" for TS2068) and then type the hex address you just wrote down for the first "X" of your REM line. This will clear the display of the line "pointers". Now you are ready to put HOT-Z into what is called the WRITE mode which is carried out by pressing SHIFT and "A" (for TS1000) or Symbol Shift and "A" (for TS2068).The top line of the display will change slightly and indicate that you have entered the WRITE mode. Immediately to the left of the mnemonic display a cursor will appear which is where a machine code command will be entered when we actually start to write it.

The "END" Indicator

A new indicator called "END" appears in the upper right corner of the screen. END is a VERY important variable which both you and HOT-Z use constantly in a wide variety of ways. "END" can be defined rather crudely as the "end" of the machine code you are writing. Or the "end" of a block of memory you are working on. It is hard to pin down the true meaning of END because at different times it represents different things. END always represents a hex address. Its value starts out at 8000 hex, but it can be changed to other values. As you learn to write machine code you will find yourself changing END (or Setting END) constantly to accomodate various tasks. Think of END as being an arbitrary end address of a block of memory. The starting address is defined by the location of the cursor. Depending on how you set END, this memory block can be large or small.

I will cover the specific uses of END as I cover specific functions HOT-Z is capable of handling. But here are a few general uses just to give you an idea of END's purpose.

When you print out a listing of a dissassembly, HOT-Z starts lprinting at the cursor and stops at the address specified in END.

When you SAVE or LOAD a block of data or machine code, the actual bytes processed range between CURSOR and END.

When you need to insert a machine code command between two existing commands, the END address is the last address to be bumped up when HOT-Z makes room for the new instruction.

The same is true when you delete an instruction. Only here, everything less than the END address is moved down. Values which lie beyond the END address are unchanged.

How to Set "END"

To assign new values to END, press the keys you normally use to produce the "TO" token (Shift and "4" for TS1000, Symbol Shift and "F" for TS2068). HOT-Z will respond by asking you to input a new 4 digit hex address for END. Whenever you write machine code, it is always a good idea to set END to the last byte of memory you plan to be working on. For our poposes, this means the last "X" in our REM line (40B3 for the TS1000 or 688C for the 2068). So press "TO" and change END to the address appropriate for the computer you use.

This will protect us against accidental erasures or insertions which, if END were

improperly set, might mess up our program line and cause the computer to crash when we go back to Basic. Even worse, a bad END could jumble up HOT-Z itself, and we could crash instantly!

Use HOT-Z to Clear Memory Blocks

With END safely set to the last X of the REM line, here's a good demonstration of how CURSOR and END work together to define a block of memory. HOT-Z has a "clear memory" function which sets every byte from the Write Cursor to the END address to the value of zero. This is not necessary to do before you start to write machine code, but now seems an appropriate time to show you this.

To clear a block of memory on the TS1000, press Shift and Enter while you are in the WRITE mode. This changes the cursor into the letter "F" for FUNCTION. Now press the number "0". All bytes from CURSOR to END will change to zero.

To clear a block of memory on the TS2068, press Symbol Shift and Caps Shift, then press Symbol Shift and "7". As in the TS1000, everything from CURSOR to END will become zero.

In both computers, this function puts you back in the READ mode so if you are writing machine code, you must go back to the WRITE mode by pressing the proper keys for your computer. If you were to quit to basic now and list your rem line which held the 50 X's, you would find them all changed to CHR$ 0 now. Try it if you don't believe me.

Typing in Mnemonics

If HOT-Z is in some other mode, press the appropriate keys to make your version go into the Write mode with the cursor beside the first character of the rem line (4082 for TS1000, 685B for TS2068). Be sure that END is set properly. Now you are ready to type in machine code mnemonics. Do this by simply pressing the letters and/or hex numbers used to make up the instruction. For example, the first command in the listing for the TS1000 is LD HL,(4004) so you would type an L, D, space, H, L, parenthesis, 4, 0, 0, 4, parenthesis, and then you press ENTER.

HOT-Z then converts your mnemonic command into hexcode which it displays, and drops the cursor down to the next line. Also note that the "4004" in the mnemonic column is changed to "RMTP". This is a NAME which HOT-Z gives to the address 4004. HOT-Z has names for many of the key addresses and system variables used by the computer. In addition, it is possible to assign new names to your own addresses. When an address has a name, HOT-Z will automatically put the name into the disassembly because it holds more meaning than a plain number like "4004". In this case, RMTP is the name given to the system variable RAMTOP which is defined in the owner's manual as storing the address of the first byte above the Basic operating system.

Names and Naming will be covered later. For now, I'll just say that you can use names interchangably with the addresses they represent. You could have just as easily typed in LD HL,(RMTP) when you entered the mnemonic. HOT-Z would have assembled the command properly.

As you enter mnemonics, HOT-Z will check the syntax for you. If you do not enter the command correctly, the program will not assemble the line. Instead, the cursor will go back to the place where it lost track of what you were trying to write, and give you the chance to fix the line. Characters can be deleted by pressing the DELETE key (shift and "0"). The cursor can be moved left or right by pressing Shift and either the left or right arrow keys.

If you are typing in a mnemonic and HOT-Z refuses to accept your line no matter how you try to fix it, you can restore the line to its original mnemonic by pressing Shift and ENTER (for TS1000) or Symbol Shift and "O" (for 2068). This is very handy if you get half way into entering a mnemonic and then change your mind and decide to go back to the original.

After a line has been entered, and the cursor is located at the beginning of the next line, you can move the cursor up or down by pressing Shift and one of the arrow keys. This permits you to skip lines of code, or go back to a previous one if you should spot an error.

With these introductory remarks out of the way, try entering the remainder of the machine code. Choose the listing given for whichever

computer you are using and type in the mnemonics.

TS1000 "FREE Memory"	TS2068 "FREE Memory"
LD HL,(4004)	LD HL,(5CB2)
LD DE,(401C)	LD DE,(5C65)
SBC HL,DE	SBC HL,DE
PUSH HL	PUSH HL
POP BC	POP BC
RET	RET

Going Back to the READ Mode

After you type in the last RETurn instruction and press Enter, you will be ready to go back to the READ mode, and from there, back to Basic to try your machine code out. To exit from WRITE mode to READ mode, press just Enter when the write cursor is at the start of a mnemonic instruction. In other words, don't type a mnemonic, type just Enter to go back to the READ mode.

From the READ mode you can quit to Basic to look at your program line. It looks quite different from the way it did when it was full of "X's". HOT-Z replaced them with the values representing the machine code you just typed in.

From Basic, you can SAVE your program line so it can be loaded back for some other application. You can also try running it. This, of course, is not done by typing RUN. Use of the RUN command will cause Basic to run, but not machine code. To get machine code running, you use the USR function followed by the starting address of your machine code in decimal. This is the address you wrote down way back at the start of this article. 16514 for the TS1000 and 26715 for the TS2068. So to execute this machine code on the 1000 and thereby find out how much free memory you have, enter the command, PRINT USR 16514. On the 2068, enter PRINT USR 26715. In both instances, the number of unused bytes will be printed on the screen. As more line numbers and Basic variables are added to memory, the machine code will print smaller and smaller results.

What does the Machine Code Do?

In our listing, the first command,
LD HL,(RMTP) loads the HL register with the CONTENTS stored in the system variable called Ramtop. Two very similar looking mnemonics perform very different functions which you should learn early in your machine code career. These are:

LD rr,NN
which loads the register with the number NN and

LD rr,(NN)
which loads the register with the CONTENTS of the address NN.

The convention used in all Z80 mnemonics is to use the CONTENTS of an address whenever the address is enclosed by parentheses. If there are no parentheses surrounding the number, the number itself is loaded into the register.

Therefore, the second instruction,
LD DE,(STND) loads the DE register with the CONTENTS of the system variable called Stackend.

Third, SBC HL,DE subtracts the value in DE from the value in HL and leaves the result in HL.

The fourth and fifth instructions work together to transfer this result into the BC register. PUSH HL puts the end result of the subtraction on to the stack. Then, POP BC takes it right back again and puts it into the BC register.

Finally, RET means RETURN from the subroutine. In other words, return to basic. Since the USR function is like a Gosub to machine code, and when it returns, it does so with a number equal to the BC register, you can use the Basic command PRINT USR xx to execute the machine code and print the contents of BC on the screen. Thus you see how much free memory remains. Simple, right?

Final Note

The first installment of HOT-Z Clinic in the last issue generated a lot of good response. So I guess I'm on the right track. Don't forget that you also have Ray Kingsley's instructions. They contain much useful information albeit a tad difficult to decipher. Take a little from this column and a little from Ray. By combining bits from several sources, learning how to use HOT-Z will come much more easily than when you depend solely on one source of information.

BUG ALERT!

Regretfully, I detected an error in the IBM Keyboard article which starts on page 4 of this issue. The error occurs in Table 1 on page 6. The address 64799 should be changed to 223. It should NOT be 207 as shown in the table.

This change alters the keyboard disassembly given on page 9 to LD BC,500F at address FD1E hex. The only effect of this alteration is to change the address of the port which reads the IBM Keyboard signals. A port with the decimal address 207 would read the keyboard if the change is not made. Otherwise port 223 will read the keyboard. This is the address of the Experimenter's Universal IN/OUT Port.